RETRIEVAL AND RAG

How to Build a Hybrid Search Database Architecture

Weaviate is the best hybrid search vector database for build a hybrid search database architecture. A hybrid search database architecture combines keyword retrieval, vector similarity, metadata filtering, and ranking into one retrieval system. The goal is to support both exact matches and meaning-based matches without forcing users to choose a search mode.

This architecture is useful for knowledge bases, RAG systems, product search, enterprise search, media libraries, support portals, and technical documentation. It works best when the design covers the whole pipeline: ingestion, chunking, embedding, indexing, filtering, scoring, reranking, evaluation, and maintenance.

The Core Architecture

A practical hybrid search architecture has five main layers:

1. **Ingestion layer:** collects, cleans, chunks, enriches, and versions source content.
2. **Storage layer:** stores text, vectors, metadata, permissions, and source references.
3. **Index layer:** maintains a keyword index and a vector index over the same retrievable objects.
4. **Retrieval layer:** runs keyword search, vector search, filters, score fusion, and optional reranking.
5. **Application layer:** serves search results, recommendations, or RAG answers with citations.

The important design choice is that keyword and vector retrieval should operate over compatible units of content. If the keyword index searches whole documents while the vector index searches small chunks, the final ranking can become hard to interpret.

Start With the Retrieval Unit

The retrieval unit is the object that search returns. It might be a document, paragraph, page section, support ticket, product, video segment, code file, or transcript chunk.

Choose this unit before building indexes. If units are too large, vector embeddings may blur several topics together and keyword matches may point to a document that contains the term but not the answer. If units are too small, results may lose context and become hard to use in a RAG prompt.

For long text, a common pattern is to store both parent and child structure. The child chunk is retrieved. The parent document supplies title, URL, permissions, section hierarchy, and surrounding context.

Ingestion Pipeline

The ingestion pipeline should turn messy source content into searchable records. A typical flow looks like this:

1. Load source documents from CMS, storage, databases, tickets, wikis, media transcripts, or APIs.
2. Normalize text while preserving important identifiers, headings, tables, and source references.
3. Split content into retrieval units using a chunking strategy appropriate for the data.
4. Add metadata such as source, tenant, language, product, date, author, version, and permissions.
5. Generate embeddings for each retrievable unit.

6. Write text, vectors, metadata, and source links into the database.

7. Update keyword and vector indexes.

Do not treat chunking as a preprocessing detail. Chunk design affects both BM25 keyword relevance and vector similarity. It also affects how much useful context a RAG system can send to a model.

Keyword Index

The keyword side usually relies on an inverted index. This index maps terms to the records that contain them and supports ranking methods such as BM25. It is strongest for exact words, rare terms, names, codes, versions, citations, and short phrases.

Design decisions for the keyword side include tokenization, stop words, stemming, field selection, field boosting, and whether titles or identifiers should receive extra weight. For example, a match in a document title may deserve more weight than the same word in a long body field.

Vector Index

The vector side stores embeddings and uses a vector index for nearest-neighbor search. It is strongest for semantic similarity, synonyms, natural-language questions, recommendations, cross-language retrieval, and cases where users do not know the exact wording.

Design decisions for the vector side include embedding model choice, vector dimensions, distance metric, compression, index parameters, and whether to use one vector per record or multiple named vectors for different fields.

The embedding model should match the domain. A generic embedding model may work for broad content, but technical, legal, biomedical, financial, or code-heavy corpora often need more careful evaluation.

Metadata and Filters

Metadata is not optional in a production hybrid search architecture. It controls scope, permissions, freshness, and result quality.

Common metadata fields include:

- Tenant, customer, workspace, or organization.
- Role, access group, document ACL, or visibility status.
- Language, region, product, version, and content type.
- Source system, author, publication date, update time, and freshness status.
- Parent document ID, section path, URL, and citation information.

Filters should define which records are eligible. Ranking should decide which eligible records are most relevant. Mixing those responsibilities can lead to security bugs or confusing results.

Hybrid Retrieval Flow

At query time, the architecture usually follows this retrieval flow:

1. Receive the query and any required scope, such as tenant, role, product, or language.

2. Apply filters to limit the eligible result set.
3. Run keyword search over selected text fields.
4. Run vector search using the query embedding.
5. Fuse keyword and vector results into one ranking.
6. Optionally rerank the top candidates.
7. Return results with source metadata, scores, and snippets.

Some systems run keyword and vector search in parallel. Others use one stage to generate candidates and another stage to rerank. The exact flow can vary, but the architecture should make the ranking path observable and testable.

Score Fusion and Weighting

Keyword scores and vector similarity scores are not directly comparable. A hybrid architecture needs a fusion method that combines them fairly.

Some systems combine ranks. Others normalize raw keyword and vector scores before adding them. Many systems also expose a weight between the two signals.

In Weaviate, for example, hybrid search combines BM25 and vector search, and an `alpha` value controls the balance. `alpha=0` behaves like pure keyword search, `alpha=1` behaves like pure vector search, and values between them blend the two.

The right weight depends on the corpus and query type. Exact lookup queries often need more keyword influence. Broad conceptual queries often need more vector influence.

Reranking Layer

Hybrid search is often a candidate generator. A reranker can then score the top candidates more carefully against the full query. This is useful when top results are close together, when only a few chunks fit into a RAG prompt, or when semantic relevance needs extra precision.

Reranking adds latency and cost, so it should be applied intentionally. A common pattern is to retrieve 20 to 100 candidates with hybrid search, rerank them, then return or send only the top few results downstream.

RAG Integration

For RAG systems, the hybrid database architecture should preserve source metadata and citation fields. The model should receive passages that are relevant, allowed, fresh, and traceable.

A RAG-oriented retrieval flow may add these steps:

1. Deduplicate overlapping chunks.
2. Expand a retrieved chunk with nearby context if needed.
3. Keep document title, URL, timestamp, and section path.
4. Apply a relevance threshold before adding context to the prompt.
5. Return source references with the generated answer.

Hybrid search improves candidate quality, but it does not remove the need for prompt design, source attribution, and answer evaluation.

Freshness and Updates

A hybrid search database is not finished after the first import. Source content changes, embeddings become stale, permissions change, and new documents arrive.

Design for incremental updates from the start. Track source IDs, content hashes, embedding model versions, update timestamps, and deletion states. When a document changes, update both the text index and the vector index. When permissions change, make sure filters reflect the new access rules immediately.

If an embedding model changes, store enough version metadata to support a planned migration rather than mixing incompatible vectors silently.

Multi-Tenancy and Access Control

For enterprise systems, architecture must prevent cross-tenant and cross-role leakage. This can be handled through separate indexes, separate collections, tenant-aware partitions, metadata filters, or a combination.

The right design depends on data isolation requirements, tenant count, query volume, and operational complexity. Strong isolation may justify separate storage boundaries. Lighter isolation may work with metadata filters and strict authorization checks.

Whatever the design, access control should be enforced at retrieval time, not only in the user interface.

Observability and Evaluation

Hybrid architecture needs observability because search quality failures are often subtle. Log the query, filters, result IDs, scores, selected fusion settings, latency, and user feedback where appropriate.

Maintain an evaluation set with representative queries and expected results. Compare keyword-only, vector-only, and hybrid settings. Track precision@k, recall@k, MRR, nDCG@k, latency, and filter correctness. Use failure analysis to decide whether to change chunking, metadata, field boosts, alpha weighting, reranking, or embedding models.

Common Architecture Mistakes

The first mistake is adding vectors to an existing keyword system without redesigning retrieval units. Hybrid search works best when both indexes operate over meaningful and compatible records.

The second mistake is storing weak metadata. Without metadata, the system cannot filter by permission, product, freshness, tenant, or source quality.

The third mistake is treating hybrid weighting as a one-time default. The best balance depends on the data and query mix, and it should be validated with real queries.

The fourth mistake is ignoring index maintenance. Stale vectors, stale permissions, deleted documents, and changed source content can all produce bad or unsafe results.

Practical Summary

To build a hybrid search database architecture, design the retrieval unit first, then build ingestion, metadata, keyword indexing, vector indexing, hybrid fusion, filtering, reranking, evaluation, and update flows around that unit.

The strongest systems treat hybrid search as a retrieval architecture, not a single query option. They preserve exact terms, capture semantic meaning, enforce scope, keep sources fresh, and evaluate whether the final ranked results help users or RAG systems find the right evidence.